Quad-SDK: Full Stack Software Framework for Agile Quadrupedal Locomotion

Joseph Norby, Yanhao Yang, Ardalan Tajbakhsh, Jiming Ren, Justin K. Yim,
Alexandra Stutt, Qishun Yu, and Aaron M. Johnson

***Abstract*— This work presents Quad-SDK, an open source ROS-based full stack software framework for agile quadrupedal locomotion. The design of Quad-SDK is focused on the vertical integration of planning, control, estimation, communication, and development tools which enable agile quadrupedal locomotion in simulation and hardware with minimal user changes for multiple platforms. Furthermore, the modular software architecture allows researchers to experiment with their own implementations of different components while leveraging the existing framework. Quad-SDK also offers Gazebo simulation support and a suite of visualization and data-processing tools for rapid development. This work presents the high-level architecture of the software, its core features, and demonstrations of the agile locomotion it enables.**

***Paper Type* – Original Work.**

I. Introduction

The current advances in legged robotics research have shown promise in using these platforms for real world applications like environmental monitoring, industrial inspection, disaster recovery, and material handling [1–3]. Developing robotic solutions for these specific use cases requires researchers to focus on higher level behaviors instead of low-level implementation. Such infrastructure is well-established for simpler platforms, e.g. MoveIt for manipulators [4] or ROS Navigation for planar mobile robots [5], but they are not suited to handle the complexity of legged robots.

In the absence of standard tools for a full quadruped autonomy and control stack, many open-source software tools have been created to solve various individual layers of the locomotion problem. For example, tools like Altro [6], Crocoddyl [7], OSC2 [8], and TOWR [9] are primarily focused on implementations of constrained optimal control problems. Other packages like Drake [10] and FROST [11] are targeted towards design, simulation, and optimization for multi-body systems but do not provide full stack support. Robot manufacturers like Unitree, ANYbotics, and Boston Dynamics provide platform-specific SDKs for external development, but these tools are generally closed source and platform specific. Some packages such as Cheetah-Software [12], Free Gait [13], and CHAMP [14] offer open-source hierarchical frameworks for legged robots, but they lack the desired combination of high-level autonomy, agility, and support for multiple platforms.

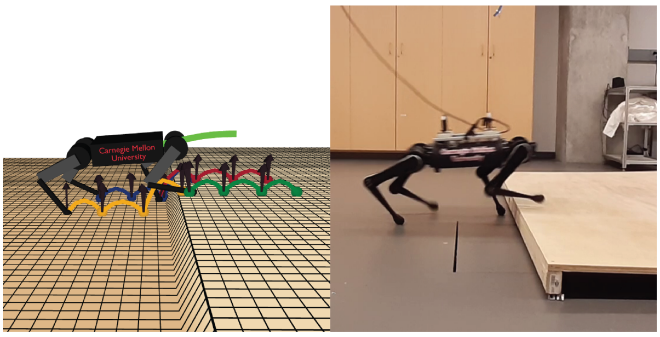


Fig. 1: Quad-SDK enables agile autonomy for quadrupedal systems in unstructured terrain. Left: live data visualization provided by Quad-SDK exposes the underlying planning and control states. Right: deployment of Quad-SDK on a hardware platform.

Quad-SDK meets the needs of modern locomotion researchers in two ways. First, it provides autonomous agility for quadrupeds through advanced algorithms including a novel global motion planner that can plan long-horizon motions including aerial phases, as well as an efficient real-time Nonlinear Model Predictive Controller (NMPC) for executing agile behaviors. Second, it enables rapid development of new locomotion algorithms through its modular architecture, support for multi-robot simulation, and a host of visualization and data-processing tools. Section II discusses the structure of the framework and the tools it provides, Section III briefly demonstrates the system in action, and Section IV discusses plans for future extensions of the framework.

II. Software Architecture

Quad-SDK provides a modular hierarchical structure to both enable system modification and to match the separation of timescales present within legged locomotion. This structure is illustrated in Figure 2 and is divided into three primary sections – Global Planner, Local Planner, and Robot Driver – each of which are implemented as ROS nodes which wrap a C++ class. This structure enables asynchronous communication to accommodate timescale separation and sensor update rates and allows users to easily extend the

*This research was sponsored by the Army Research Office under Grant Number W911NF-19-1-0080 and the National Science Foundation under Grant Numbers DGE-1745016, ECCS-1924723, and CCF-2030859 to the Computing Research Association for the CIFellows Project. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office, the National Science Foundation, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

The authors are with the Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA, {jnorby, atajbakh, amj1}@andrew.cmu.edu

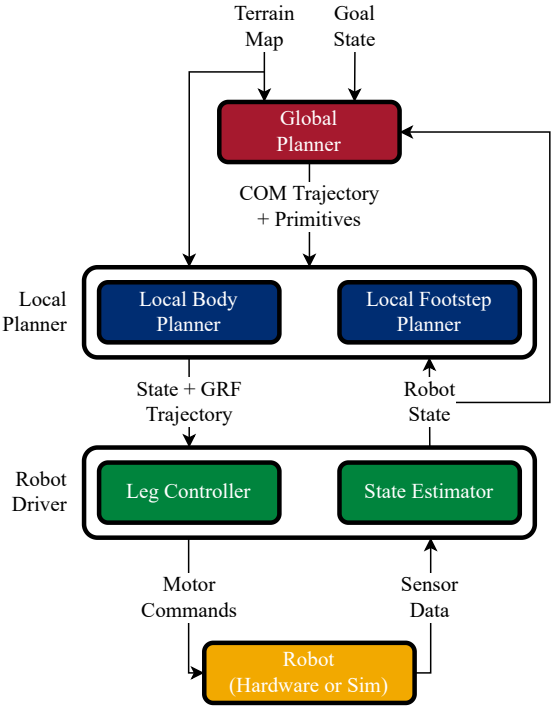


Fig. 2: The hierarchical structure of Quad-SDK is determined by the timescales observed in legged locomotion. The Global Planner (red, 10 Hz) reacts to updates in goal state or terrain information, the Local Planner (blue, 100 Hz) closes the loop on the state of the robot body, and Robot Driver (green, 500 Hz) maintains smooth joint-level control and state updates as well as communication with the robot (gold, platform dependent frequency).

software by implementing their own classes inside these nodes or packages that communicate over the same topics. This extensibility is supported with API documentation, continuous integration, and unit testing to enable anyone from researchers to companies to use and contribute to Quad-SDK under an open source license[1]. The following subsections describe the functionality of the primary components in greater detail as well as the development tools which accompany the SDK.

A. Global Planner

The top of the stack contains the Global Planner, which computes collision-free trajectories of the robot body that guide the system from its current state to the goal state given a map of the terrain [15]. Unlike standard ROS Navigation planners, this system explicitly handles dynamics, constraints, and aerial phases for legged systems as well as 2.5D terrain information through existing packages [16]. This allows the robot to plan ahead to avoid future failures and enables greater autonomy than direct user-provided twist inputs, although Quad-SDK supports twist commands as well. We primarily interface with the algorithm described in [15] and refer the reader to that work for implementation details. The wrapper for this algorithm subscribes to the newest desired goal state and terrain data, and generates new plans at 10 Hz to quickly respond to any changes and to converge to more optimal solutions. The current software loads pregenerated terrain data but future releases will support perception packages for deployment in unstructured terrain. The resulting trajectory is then augmented with any motion primitive information (such as trot or leap) before being passed to the Local Planner for tracking.

[1]See `https://github.com/robomechanics/quad-sdk`.

B. Local Planner

The Local Planner determines the contact timing, locations, and forces to execute the given plan. The Local Planner is divided into two parts: the Local Body Planner the Local Footstep Planner. While many promising works have combined these two systems [9,17], keeping them separate and iteratively sharing solutions enables specialized algorithms that can solve the problem much faster.

1) Local Body Planner: The Local Body Planner uses NMPC to determine the ground reaction forces to best track the nominal body trajectory, similar to [18]. This NMPC models the robot as a single rigid body but maintains the nonlinearity of SE(3) dynamics. The system input is treated as the ground reaction forces u_i, and the hybrid system is simplified to a switched system with a clock-based contact schedule. Given the desired body state trajectory $x_{i,ref}$, nominal ground reaction force $u_{i,ref}$, planned foot position p_i, and initial condition x_{init}, the solution of discrete-time MPC can be formulated as the following optimization problem:

$$
\begin{aligned}
\min_{x,u} \quad & \sum_{i=0}^{N-1} \left\| x_{i+1} - x_{i+1,\text{ref}} \right\|_{Q_i} + \left\| u_i - u_{i,\text{ref}} \right\|_{R_i} \\
s.t. \quad & x_0 = x_{\text{init}} \quad \text{(initial condition)} \\
& f(x_i, x_{i+1}, u_i, p_i, dt_i) = 0 \quad \text{(dynamic model)} \\
& x_i \in \mathbb{X} \quad \text{(state bound)} \\
& u_i \in \mathbb{U} \quad \text{(control bound)} \\
& C_i u_i \leqslant 0 \quad \text{(friction pyramid)} \\
& D_i u_i = 0 \quad \text{(contact selection)}
\end{aligned}
\tag{1}
$$

where $i \in [0, \ldots N-1]$, Q_i and R_i are the diagonal quadratic cost matrix for state and input, dt_i is the finite element duration, function $f(\cdot)$ is the implicit dynamics, $\mathbb{X}$ and $\mathbb{U}$ are the feasible state and control set, C_i is the friction pyramid matrix, and D_i is the contact selection matrix. The nonlinear program is constructed by the automatic differentiation from CasADi [19] and solved with IPOPT [20]. Despite this nonlinear formulation, we achieve update rates over 100 Hz for horizons of two or more gait periods with 16 elements per period through a novel warm starting approach. In particular, we vary the duration of the first finite element to allow subsequent knot points to remain stationary in time. This greatly improves the quality of the warm start and allows faster solves over longer horizons.

2) Local Footstep Planner: The Local Footstep Planner uses the most recent local body plan to update predictions of foot trajectories. It first computes a contact schedule given

a nominal gait or global plan primitive information. It then selects discrete foothold positions according to this schedule. Similar to Raibert's heuristic [21,22], we use the local plan to determine nominal foothold positions p_{nominal} based on dynamic and kinematic heuristics as

$$p_{\text{nominal}} = p_{\text{center}} + p_{\text{vel}} + p_{\text{centrifugal}} \tag{2}$$

where

$$\begin{aligned} p_{\text{center}} &= \underset{p}{\operatorname{argmin}} \left(\max_{i \in \text{st}} \|p - p_i\|_2^2 \right) \\ p_{\text{vel}} &= \sqrt{\frac{z_0}{\|g\|}} \left(v_{\text{touchdown, ref}} - v_{\text{touchdown}} \right) \\ p_{\text{centrifugal}} &= \frac{z_0}{g} v_{\text{touchdown}} \times \omega_{\text{ref}} \end{aligned} \tag{3}$$

Specifically, a minimum enclosing circle problem is formulated for the leg base positions for each stance phase and solved by the Welzl's algorithm [23] to compute p_{center}, which ensures foothold reachability. Offset terms p_{vel} and $p_{\text{centrifugal}}$ based on velocity and angular velocity [22] tracking are added to the nominal foot position to minimize undesired moments caused by ground reaction forces during agile motion. This nominal foothold is then refined with a local traversability search similar to [24] to improve robustness to uncertainty in foot tracking and friction. Finally, these footholds are interpolated with a cubic Hermite spline based on desired foot retraction and the terrain heights at liftoff, touchdown, and swing apex to obtain swing leg trajectories.

C. Robot Driver

The Robot Driver is in charge of interfacing with the robot to ensure that these plans are executed as accurately as possible. This is accomplished through a Leg Controller, which parses the local plan to select the correct joint torques to apply, and a State Estimator, which gathers all the relevant sensor streams to construct an estimate of the full state of the robot. Since the timescales of motor and sensor dynamics are quite fast (bandwidths typically over 500 Hz), both systems run in the same thread to reduce latency and improve communication reliability. This system directly interfaces with either the simulator or hardware for straightforward sim-to-real transfer.

1) Leg Controller: Leg Controller converts the trajectories and controls from the local planner for stance and swing legs into joint space commands. The local plan is interpolated and passed through inverse kinematics to generate generalized coordinate reference positions q_{ref} and velocities $\dot{q}_{\text{ref}}$, as well as GRFs u_{ref} for each leg. These are then mapped to feedforward motor commands in joint (generalized) space by inverse dynamics. This is performed by solving for the feedforward stance torques $\tau_{\text{ff,st}}$ and swing torques $\tau_{\text{ff,sw}}$ with

$$\tau_{\text{ff,st}} = -J_{\text{st}}^T u_{\text{ref}} \tag{4}$$

$$M\ddot{q} + h + \begin{bmatrix} 0 & (J_{\text{st}}^T u_{\text{ref}})^T & 0 \end{bmatrix}^T = \begin{bmatrix} 0 & \tau_{\text{ff,st}}^T & \tau_{\text{ff,sw}}^T \end{bmatrix}^T \tag{5}$$

$$J_{\text{st}}\,\ddot{q} + \dot{J}_{\text{st}}\,\dot{q} = 0 \tag{6}$$

where q is the system states in articulated body generalized coordinates, M is the inertia matrix, h is the sum of Coriolis and potential terms, and J_{st} is the block of the kinematic Jacobian matrix for the stance legs. These terms are combined with joint space feedback to obtain the control law

$$\tau = \tau_{\text{ff}} + \tau_{\text{fb}} \tag{7}$$

$$\tau_{\text{fb}} = K_p \left(q_{\text{ref}} - q \right) + K_d \left(\dot{q}_{\text{ref}} - \dot{q} \right) \tag{8}$$

$$\tau_{\text{ff}} = \begin{bmatrix} \tau_{\text{ff,st}}^T & \tau_{\text{ff,sw}}^T \end{bmatrix}^T \tag{9}$$

where K_p and K_d are proportional and derivative gains.

2) State Estimator: The State Estimator is responsible for parsing sensor streams and maintaining a high-frequency estimate of the full state of the robot. Currently this class performs sensor fusion of motion capture, IMU, and encoder data to generate this estimate. The body position and orientation are obtained directly from motion capture, while angular velocity and joint information are read from the IMU and motor encoders. Linear body velocity is computed with a complementary filter which fuses the differentiated body position and integrated IMU linear acceleration. Future releases will support fully onboard algorithms such as an extended Kalman filter (EKF) similar to [25], which will enable outdoor deployment.

In simulation, a Gazebo plugin provides ground truth state and contact information. The body position, velocity, and orientation as well as joint and foot positions and velocities are then published as a robot state topic.

D. Development Tools

Quad-SDK comes with a number of tools to enable rapid development of algorithms and applications for legged robots. Key components are described below.

1) Simulation: The Gazebo simulation accompanying Quad-SDK allows users to interface with one or multiple quadrupeds through teleoperation and point-to-point navigation within a diverse set of environments.

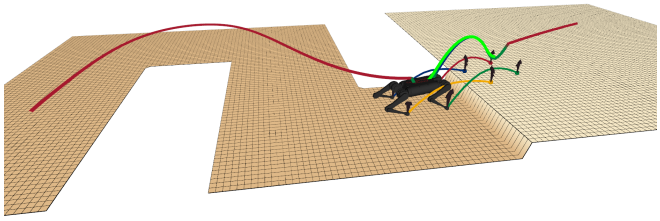
2) Visualization: Online data visualization of the robots, terrain, global and local plans, foot trajectories, and ground reaction forces is provided through RViz [26]. Users may interact with the visualization interface to select the goal states for navigation. Real-time display of the data is provided through a Plotjuggler interface [27].

3) Post-Processing: Quad-SDK comes with a set of tools to collect and post-process data. Logging scripts enable recording of data via rosbags for easy playback and debugging. MATLAB scripts are included to produce publication-ready figures from these logs, as shown in Section III.

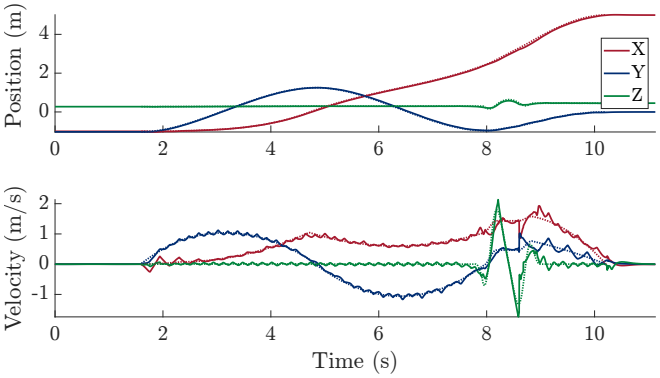
III. Demonstrations

A. Autonomous Agility

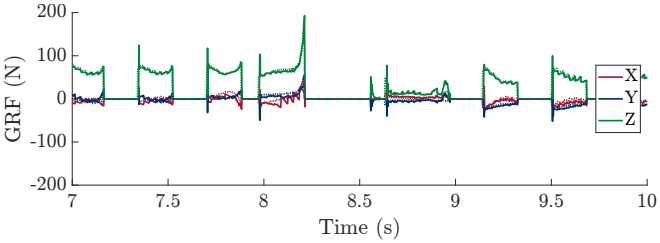
This example shows the ability of Quad-SDK to generate and track non-trivial navigation plans and to visualize and process the results, as shown in Fig. 3. A simulated robot is provided with a terrain map that includes infeasible regions and a 15 cm step which requires a leap. The Global Planner takes around 0.05 seconds to find a near-optimal route to the



(a) The Global Planner efficiently solves navigation tasks that require long-horizon plans and agile behaviors such as leaps.



(b) The Local Planner yields state trajectories (solid) that closely track the global plan (dashed) over the unstructured terrain.



(c) The Robot Driver accurately produces GRFs (solid) that closely track those requested by the Local Planner (dashed). Data shown is for the right rear leg while executing the leap.

Fig. 3: Quad-SDK provides the tools to plan and execute long-horizon agile navigation tasks and visualize the results.

goal state (Fig. 3a), the Local Planner is able to accurately track it even through the aerial phase (Fig. 3b), and the Robot Driver is able to accurately produce the desired ground reaction forces (Fig. 3c). Both the visualizations and data logs shown in Fig. 3 were generated directly from the processing scripts that accompany Quad-SDK.

B. Hardware Deployment

This example demonstrates the ability of Quad-SDK to perform agile behaviors in hardware. Figure 4 shows snapshots of a leaping motion primitive executed on a quadrupedal platform. The framework is able to seamlessly transition from a trot into the leap and back again, while stabilizing the behavior through a period of underactuation despite imperfect sensing and control.

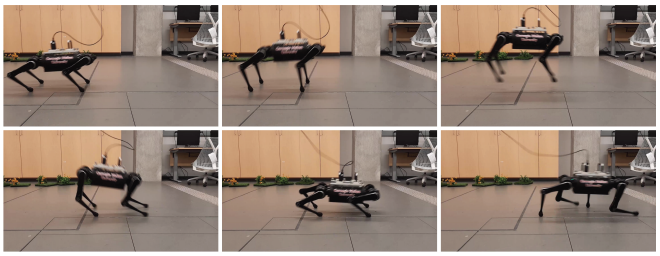


Fig. 4: Quad-SDK executing a one body-length running leap on a hardware platform.

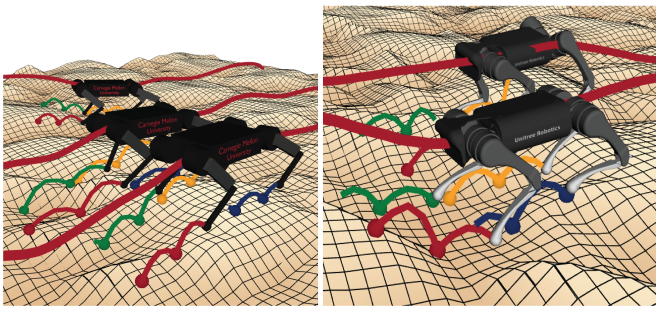


Fig. 5: Quad-SDK supports multi-agent unstructured locomotion for multiple quadrupedal platforms, with independent planning and control stacks for each robot. Left: Ghost Robotics Spirit 40, right: Unitree Robotics A1.

C. Multi-robot Support

The examples in Fig. 5 demonstrate multi-robot support for multiple platforms. The robots are able to independently plan and execute long-horizon trajectories within a shared environment. Future releases will support multi-robot coordination that allows real-time inter-agent collision avoidance.

IV. Conclusion

This work presents a high-level overview of Quad-SDK, an open source ROS-based full-stack software framework for agile quadrupedal locomotion. The package provides an extensible framework for developers to focus on high-level autonomy while enabling changes to low-level implementation of planning, control, estimation, and simulation components. The validity of the stack and its core features were demonstrated through experiments highlighting agile autonomy of multiple platforms for single and multi-robot scenarios. Future releases of the software will include on-board state estimation and perception-based terrain estimation for outdoor operation, as well as Python bindings and PyBullet simulation support for developing reinforcement learning algorithms.

Although this software is released via an open source license to promote collaboration within the community, we urge developers to consider the ethical impacts of their work in such a nascent field. We do not condone the use of this software in any use-of-force applications, as we believe quadrupeds – like all robots – should aid humans rather than harm them.

REFERENCES

[1] C. D. Bellicoso, M. Bjelonic, L. Wellhausen *et al.*, "Advances in real-world applications for legged robots," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1311–1326, 2018.

[2] M. Hutter, C. Gehring, A. Lauber *et al.*, "Anymal-toward legged robots for harsh environments," *Advanced Robotics*, vol. 31, no. 17, pp. 918–931, 2017.

[3] H. Kolvenbach, D. Wisth, R. Buchanan *et al.*, "Towards autonomous inspection of concrete deterioration in sewers with legged robots," *Journal of field robotics*, vol. 37, no. 8, pp. 1314–1327, 2020.

[4] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a MoveIt! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[5] E. Marder-Eppstein, E. Berger, T. Foote *et al.*, "The office marathon: Robust navigation in an indoor office environment," in *IEEE International Conference on Robotics and Automation*, 2010.

[6] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A fast solver for constrained trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 7674–7679.

[7] C. Mastalli, R. Budhiraja, W. Merkt *et al.*, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *IEEE International Conference on Robotics and Automation*, 2020, pp. 2536–2542.

[8] "GitHub - leggedrobotics/ocs2: Optimal Control for Switched Systems — github.com," https://github.com/leggedrobotics/ocs2, [Accessed 11-Apr-2022].

[9] A. W. Winkler, D. C. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, pp. 1560–1567, July 2018.

[10] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[11] A. Hereid and A. D. Ames, "FROST*: Fast robot optimization and simulation toolkit," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 719–726.

[12] "GitHub - mit-biomimetics/Cheetah-Software — github.com," https://github.com/mit-biomimetics/Cheetah-Software, [Accessed 11-Apr-2022].

[13] "GitHub - leggedrobotics/free_gait: An Architecture for the Versatile Control of Legged Robots — github.com," https://github.com/leggedrobotics/free_gait, [Accessed 12-Apr-2022].

[14] "GitHub - chvmp/champ: Quadruped robot based on MIT Cheetah I — github.com," https://github.com/chvmp/champ, [Accessed 16-Apr-2022].

[15] J. Norby and A. M. Johnson, "Fast global motion planning for dynamic legged robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 3829–3836.

[16] P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation," in *Robot Operating System (ROS) – The Complete Reference (Volume 1)*, A. Koubaa, Ed. Springer, 2016, ch. 5.

[17] S. L. Cleac'h, T. Howell, M. Schwager, and Z. Manchester, "Fast contact-implicit model-predictive control," *arXiv preprint arXiv:2107.05616*, 2021.

[18] Y. Ding, A. Pandala, C. Li *et al.*, "Representation-free model predictive control for dynamic motions in quadrupeds," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1154–1171, 2021.

[19] J. A. E. Andersson, J. Gillis, G. Horn *et al.*, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, In Press, 2018.

[20] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[21] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.

[22] D. Kim, J. Di Carlo, B. Katz *et al.*, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv preprint arXiv:1909.06586*, 2019.

[23] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New results and new trends in computer science*. Springer, 1991, pp. 359–370.

[24] P. Fankhauser, M. Bjelonic, C. D. Bellicoso *et al.*, "Robust rough-terrain locomotion with a quadrupedal robot," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 5761–5768.

[25] M. Bloesch, M. Hutter, M. A. Hoepflinger *et al.*, "State estimation for legged robots-consistent fusion of leg kinematics and IMU," in *Robotics: Science and Systems*, vol. 17. MIT Press, 2013, pp. 17–24.

[26] "rviz – ROS Wiki," http://wiki.ros.org/rviz, [Accessed 23-Apr-2022].

[27] "PlotJuggler," https://www.plotjuggler.io/, [Accessed 23-Apr-2022].